North-Holland
Microprocessing and Microprogramming 27 (1989) 77–84

A knowledge-based approach to VLSI-design in an open CAD-environment

L.K. Alberts, C. Huijs, N.J.I. Mars and L. Spaanenburg*
University of Twente, Department of Computer Science,
Postbus 217, 7500 AE Enschede, The Netherlands

Abstract

A knowledge-based approach is suggested to assist a designer in the increasingly complex task of generating VLSI-chips from abstract, high-level specifications of the system. The complexity of designing VLSI-circuits has reached a level where computer-based assistance has become indispensable. Not all of the design tasks allow for algorithmic solutions. AI techniques can be used, in order to support the designer with computer-aided tools for tasks not suited for algorithmic approaches. The approach described in this paper is based upon the underlying characteristics of VLSI design processes in general, comprising all stages of the design. A universal model is presented, accompanied with a recording method for the acquisition of design knowledge - strategic and task-specific - in terms of the design actions involved and their effects on the design itself. This method is illustrated by a simple design example: the implementation of the logical EXOR-component. Finally suggestions are made for obtaining a universally usable architecture of a knowledge-based system for VLSI-design.

1 Characteristics of VLSI-design

Given a high-level, formal specification of the system, the designer of a VLSI-circuit is confronted with the problem of generating a description of the circuit that can be physically realized. Since this can not be done in a single design step, a stepwise refinement of the original description is carried out. For each design step the designer must choose among a set of alternative design actions suited for the situation. After each refinement verification of the results is needed and often some sort of optimization is performed. Depending on whether the results meet the necessary requirements, the designer will continue by selecting the next action or else he will have to reconsider the choices made. This cycle is repeated over and over, leading to a gradual refinement of the design.

There are several constraints acting on the design process. On the one hand the technological and physical conditions constrain the possible design actions, as a consequence of the technology available and the characteristics of the silicon respectively. Secondly, there are a number of technical specifications that have to be met, like for example the maximum size of the final chip. The technological and physical implications are primarily bottom-up, whereas the techical specifications work more top-down through the design process. So in effect, of only one the exact nature is known until they meet somewhere in the middle.

In other words: the design process can be characterised as a complex search-process - through the intermediate design descriptions that are generated - under a number of constraints. The search is not for a single best solution, but for an acceptable compromise from a range of solutions that satisfy the requirements up to a certain degree. The complexity of this process is due to (among other things):

- a very large search-space, caused by the long trajectory from high-level system specification to low-level physical description and the large number of possibilities to travel this trajectory.
- the fact that no decisive criteria are available for the evaluation of intermediate results, especially during the initial phases of the design process. A closely related problem is the lack of sufficient quantitative information in early stages, needed to determine if the technical requirements are met.
- the need to compromise between conflicting requirements on time, power dissipation and area of the design.

So the design process presents a rather under-constrained and partly ill-defined problem, which involves a lot of

* Currently attached to IMS, Stuttgart (West-Germany)

qualitative information and heuristic reasoning. Since this prevents the use of algorithmic approaches, AI techniques are introduced to guide the search-process with heuristic knowledge. Another motive for using AI for the development of CAD tools is the possibility of re-using scarce and expensive expert-knowledge. By compiling the experience of an expert-designer in a knowledge based system, others - perhaps less experienced - may benifit from this knowledge as well.

2 Recording of design choices based on a universal model of the design process

2.1 The need for a (universal) model

In order to acquire the knowledge mentioned in the previous section a model of the design process is needed. Such a model should allow for the explicit specification of design knowledge. It should also provide a framework for describing design actions and their effects on a partially refined design, and the structuring of design knowledge.

From the point of view of software development - be it conventional or AI-oriented - this framework should be as universally valid as possible, allowing for the description of different design processes. This way a lot of re-modeling of the process, restructuring of data and knowledge bases and redesigning of the software for each different situation is avoided. Furthermore it becomes possible to compare different but similar approaches to VLSI-design in terms of the strategies used and design actions carried out.

Along a similar line of reasoning it is concluded that the model should include all stages of the design process. This makes it easier to interface the different software-tools and allows for the unearthing of design-strategic knowledge. Instead of regarding a solution path through the search space as a more or less arbitrary sequence of independent design tasks, differences in the way the seperate design actions are chained are considered vital for the quality of the resulting chip. So apart from knowledge about the seperate tasks, knowledge concerning the order in which these tasks should be carried out is acquired and stored too.

In short: the model should provide a basic vocabulary for relating the knowledge underlying the choices made, to design actions and the resulting modifications in the design. Keywords in this context are *orthogonality* and *hierarchy*. Orthogonality of the set of entities used in the model is required to prevent ambiguities; there should be only one possible interpretation for each resulting description of a design or design action. There is a natural hierarchy among the different stages in the design process. This hierarchy should also show in the structuring of the design data and knowledge.

2.2 A model for the representation of VLSI-design processes

'The design' is an abstract entity, that is gradually given a concrete form in the course of the design process. Especially in the case of VLSI-design it consists of a set of descriptions only. Since a design is represented by its description(s), the obvious way to describe design actions is as transformations between descriptions. As we saw in section 1 a number of conditions and requirements are constraining the design actions. These constraints find their expression in the values of the parameters of the design. Here we choose time, energy and area as primairy parameters for a design [4].

2.2.1 Different levels of abstraction

Because the high-level specification of a system represents a very high abstraction of the low-level physical description, a number of intermediate *levels of abstraction* is introduced [1][2][3]. At each level a different model or abstraction of the physical reality is used, represented by a different representation language for describing the design. These levels form a hierarchical set, of which the number may vary depending on the characteristics of a design process. Physical and technological constraints manifest themselves in the vocabulary of the representation language available for a certain level of abstraction.

It is important to notice that by going from one level of abstraction to another a translation between two different models of the physical reality is required. Translating a description to another level of abstraction in general entails an one-to-many, instead of an one-to-one, mapping. If, for example, a circuit of NAND-gates, flip-flops and latches (logic level) is to be mapped onto a structure of transistors, capacitors and resistors (transistor level) a number of alternative configurations are possible. Notice that in the process of gradually descending in the direction of the physical level, the value of the parameters of the design become more absolute, more quantitative.

2.2.2 Views towards a design

A fundamental problem in chip design is the mapping of the functions a chip is supposed to supply onto a (basically two-dimensional) geometry of the chip. This involves a mapping of operations and their sequence onto a geometrical configuration of physical components ('blocks') with the required functionality, which depends on the properties of the silicon used. Usually an intermediate representation is generated [1] consisting of symbolic components that implement the functions and

are known to have a physical counterpart in the geometry. The resulting description is known as the structure of a design.

Hence, the following entities may be distinguished [1][4]: the function, the structure and the geometry of a design. In [2] and [4] the term 'physical domain' is used instead of geometry. This is somewhat confusing since a high-level description of the geometry of a design only describes a very rough estimation of the final physical layout. Here, the following definitions are used:

- the *function* of a design specifies the computations to be carried out and the communication of data between computations.
- the *structure* represents the circuit of symbolic components implementing the function and specifying the relative connections with the surroundings.
- the *geometry* of a design consists of the layout of physical blocks realizing the structure and the actual connections with the surroundings.

These entities define different aspects of the same design, so each design consists of a triple of descriptions: its function, structure and geometry. To emphasis this fact these entities will be called *views* [4] towards a design. They are treated as clearly distinguished, non-overlapping entities.

The interpretation of each of the design parameters differs with the view that is considered [5]. For example, time has the meaning of sequencing of operations in the function, plays a part in the timing and synchronizing of components in the structure, and has absolute properties in the geometry of a design. Area and energy hardly play any part when determining the function but are of major importance in the structure and geometry.

	Function	Structure	Geometry
Architectural level	system specs	CPU, memories	physical partitions
Register transfer level	register-transfer operations	ALU, registers,	floorplan
Logic level	boolean functions	gates, flip-flops	coarse layout
Circuit level	differential equations	transistors, capacitors, resistors	symbolic layout
Physical level	diffusion equations	cross-tacts, conductors	boxes, polygons

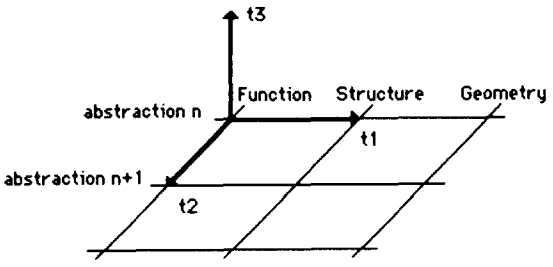
Figure 1: Levels of abstraction, views and possible representation elements (adapted from [3])

2.2.3 An orthogonal set of dimensions defining the representation space

It is possible to make the same distinction between the function, the structure and the geometry of a design (take a different view) at every stage of the design process. This also implies that all three views can be described independently of the current level of abstraction. The divisions in levels of abstraction, characterized by the way a design is represented, and in views, describing certain aspects of a design, are regarded orthogonal. We will call these divisions *dimensions* of the *representation space*. Descriptions of a design can now be characterized by their coordinates in the representation space.

In order to express the orthogonality in the way the different views towards a design are represented, for each view at a certain level of abstraction a unique set of representation elements is to be distinguised (see figure 1). Syntactically these sets may be identical, as long as their semantical differences are still recognized. Like with translations between levels of abstraction one-to-many mappings are involved going from one view to another. For instance: the calculation $x = a + b + c + d$ may be implemented as a structure consisting of one four-inputs adder or three two-inputs adders.

For our model to comprise these one-to-many mappings in an unambigious fashion as well, a third dimension is introduced orthogonal to the other two. At a certain level of abstraction and within a certain view a design can be further refined, resulting in a more detailed description, or an alternative description may be possible. Both situations will be refered to as *detailing*. The amount of detail added is totally depending on the set of representation elements in use at a certain cross-section of a view and a level of abstraction. However, the possibility for detailing as such is independent of this set and is orthogonal to the two dimensions mentioned above. Detailing, as defined here, is not the reverse process to abstracting! Abstracting in our definition involves the way the physical reality is modeled. Within such an abstraction a description may be described in more or less detail. So, for example, by rewriting our adder problem as $x = ((a + b) + (c + d))$ we add more detail to the original description. It is necessary to distinguish between this description and the original, because the extra information specified may have consequences for the implementation. Suppose we are to use three two-input adders to implement our addition. It will make quite a difference whether these are circuited semi-parallel according to the detailed function above, or in serial, giving $x = (((a + b) + c) + d)$ as the detailed function. In the second case the 'critical path' between input and output in the structure will be longer than in the first (3 resp. 2 adder components)



t1: transformation between views
t2: transformation between levels of abstraction
t3: detailing transformation

Figure 2: The representation space and the basic transformations

2.2.4 A set of basic transformations

Design actions have to be translated into transformations between descriptions. Since there are three dimensions defined for the representation space, three primairy transformation actions can be recognized. These *basic transformations* are: transformation between levels of abstraction, transformation between views and transformation based on a difference in the amount of detail. They don't specify which points in the representation space are passed, only what kind of transformation is carried out. So, in order to translate a design action in relation to the representation space, besides the transformations needed, the abstraction-levels and the views involved have to be specified (see figure 2).

Because the set of basic transformations is orthogonal, all design actions can be described explicitly and unambiguously in terms of the dimensioning of the representation space.

2.3 Recording of design choices and the underlying knowledge

The model presented in the previous section is intended as a means for acquiring the knowledge underlying the choices made during the design process. In particular, we would like to gain insight in strategic considerations concerning the order in which the different design tasks are carried out. In this context, the one-to-many mappings mentioned in 2.2.1 and 2.2.3 deserve further examination. In determining what choices have been made, by translating these mappings to the (one-to-one) basic transformations involved, the underlying design knowledge can be made explicit.

The basic mechanism for administrating the mappings consists of the introduction of 'black-box' descriptions and splitting the mappings in two. This black-box symbolizes the - possibly virtual - equivalent of the input description at the destination view or level of abstraction. One part of a mapping contains the transformation between the two views or levels of abstraction involved, resulting in a black-box description. The second part consists of a set of detailing transformations, generating the possible alternatives for detailing the black-box description. Once these alternatives have been determined, the original (input) description has to be detailed to the same amount. This is necessary to avoid inconsistencies, for example between the descriptions of the function and the structure of a design. The changes in or extensions to the original description reflect the adaptions to the design based on the constraints within the destination view or abstraction level. For instance, changes in the structure of a design may influence the functionality as well. It is important to make sure that the descriptions constituting a design relate to the same design under all circumstances. Otherwise ambiguities or even deviations from the intended design may be introduced.

2.4 An example of the recording method

To illustrate the method described in section 2.3 a simple example design is elaborated. The logical EXOR component is fundamental to the efficient design of datapaths in an integrated digital system. The possible realizations of the EXOR greatly differ in the amount of gates required. The use of socalled *Path-Driven Restoring logic* may lead to denser and faster circuitry compared to *Fully-CMOS logic* [9]. It is possible in CMOS technology to utilize the switching behaviour of MOS-transistors to implement logic structures. Basically a MOST is a bi-directional switch, but its electronic virtues strongly depend on relations between the applied signal levels. Figure 2 summarizes the possible modes of operation. Next to a closed and an open mode, a semi-open mode is available. In this mode the device displays a high output conductance together with potential signal degradation, giving rise to the socalled *weak levels*, requiring buffering of the output. In the case of Fully-CMOS logic this last mode of operation is forbidden. However, by allowing the use of the semi-opened mode, as in Path-Driven Restoring logic, the transistor count of the switching network may be optimized. To compare both technologies it is assumed that a *standard cell* design style based on a *gate-array* cell architecture is employed. This means that the degree of freedom to make changes is restricted compared to full-custom design, generally leading to a larger physical cel.

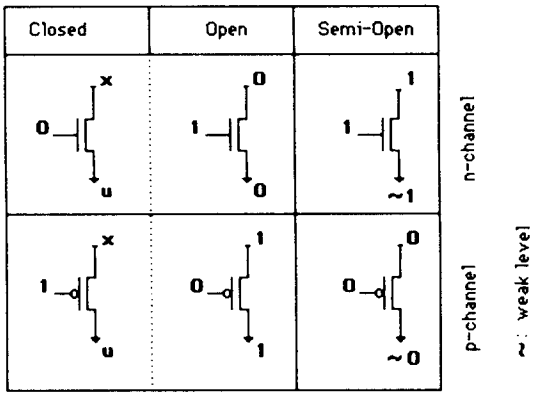


Figure 3: Modes of operation for CMOS transistors

Because this example only aims at illustrating the recording method proposed in this paper, we will not elaborate all possible realizations of the EXOR. However, in the course of describing the subsequent stages in the design relevant choices made will be pointed out. Another restriction is that the geometry of the design is not explicitly described in this case. Important geometrical constraints are accounted for by translating them into requirements on the accompaning function or structure. Finally, we will distinguish between three different levels of abstraction only: system (specification), logic and transistor level.

Suppose the designer starts with a functional description at the architectural level. One way to describe the behaviour of an EXOR is: $z = (a \neq b)$, where a and b only take digital values. The structural description might consist of a single symbolic component representing the required behaviour. Basically the designer can choose between two options for mapping the descriptions at the system level onto the transistor level: (1) via intermediate descriptions at the logic level or (2) directly.

2.4.1 Case 1: Implementation of the EXOR based on 2-input NANDs

Let's assume converting the functional description on the system level into a description on the logic level in the same view results in:

(1) $z = (\neg a \wedge b) \vee (a \wedge \neg b)$

This entails a first choice; description (1) is one out of a set of possible translations of the functional description from system to logic level. The accompaning structure is drawn in figure 4, showing a literal mapping of operations on structural (logic) components. In terms of the model proposed this is a transformation between views. The resulting structure serves as a black-box description. The implementation that is chosen is considered the result of a detailing transformation of this description. Here, the restriction made is that only 2-input NAND-gates are to be used for the final implementation, so a structural constraint. The consequence of which is, that only operations of shape $\neg(x \wedge y)$ are to be implemented.

If the functional description is taken as a starting point, application of the rules of Boolean algebra combined with a search for a minimal description results in:

(2) $z = \neg(\neg(a \wedge \neg(a \wedge b)) \wedge \neg(b \wedge \neg(a \wedge b)))$

A transformation between views results in the structure shown in figure 5a.

An alternative implementation can be obtained by starting off with the structure. If this structure is to be converted using only NANDS, the problem may be decomposed in converting the separate components that constitute the structure. In this case the logic components *not*, *and* and *or* are involved. Applying Boolean algebra to function (1) yields:

(3) $z = \neg(\neg(\neg(a \wedge 1) \wedge b) \wedge \neg(a \wedge \neg(b \wedge 1)))$

Comparing the resulting structure (figure 5b) with the first solution, shows that an extra NAND-gate is needed. Even without knowledge about the geometry, this seems likely to result in a less desirable final layout. So the first option is selected for further refinement.

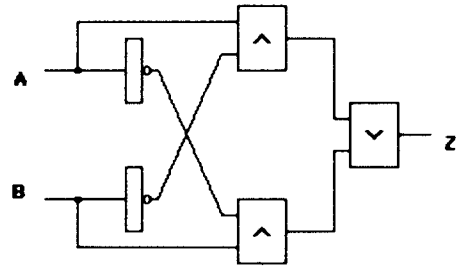


Figure 4: Logic structure of EXOR before detailing

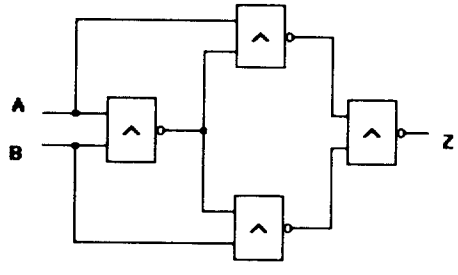


Figure 5a

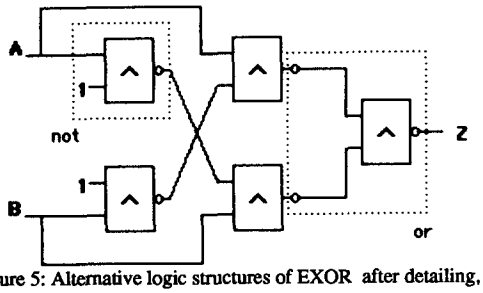


Figure 5: Alternative logic structures of EXOR after detailing, a) resp. b)

For the translation to the transistor level first a transformation between levels of abstraction is needed. This yields an intermediate description - like in the black-box approach - representing the possible translations of the logic description. Secondly a detailing transformation is required reflecting a choice from the set of alternatives. If the logic structure is to prevail, the problem reduces to implementing the behaviour of a NAND using CMOS-transistors. The functional description is omited here to avoid too much detail. The generally accepted structure of a NAND based on CMOS transistors is shown in figure 6. The structure of the EXOR at the transistor level is obtained by circuiting the transistor-equivalents of the NANDs according the logic structure in figure 5a. This approach may be characterised as using the detailed description of the structure at the logic level as intermediate (black-box) description at the transistor level.

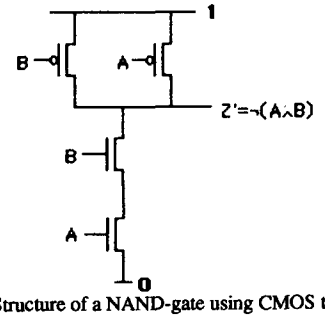


Figure 6: Structure of a NAND-gate using CMOS transistors

2.4.2 Case 2: Implementing the EXOR using the switch-logic properties of the CMOST

In the case of path-driven restoring logic, the descriptions of the EXOR at the system level are converted directly to the transistor level. As explained in the introduction to this section, this can either be accomplished using strong signal levels only or by allowing possibly degraded signal levels as well. Since the inputs only take digital values, they can be used for obtaining the required levels, '0' and '1' in figure 6, at the same time.

Starting with the system level functional description, a transformation to the transistor level is carried out. The truth table of the EXOR can serve as the required intermediate description. At this point it is important to mention some physical implications working bottum-up through the design trajectory. Since we are designing a standard cell implementation of the EXOR, it is necessary to guarantee strong output levels for our cell. In the previous situation (case 1) this was provided implicitly by the design method. In this case, however, all outputs have to be buffered to meet the requirements. This way the weak signal levels in the semi-opened mode of operation are accounted for as well. Recording this knowledge explicitly narrows the search space and prevents unnecessary iterations.

Thus we are confronted with the task of designing a circuit of CMOS transistors implementing the desired behaviour, the output of which should be connected to a buffer-circuit. A buffer in CMOS is made up of two invertors in series, each invertor requiring two transistors (see figure 7). To reduce the amount of transistors, we might as well implement the inverse of the behaviour specified. The output of this circuit is then connected to a single invertor. The combination exhibits the necessary buffering capacities and reduces the amount of transistors by two (for each buffer). Instead of generating this solution after a lengthy search process with many iterations, one might add a little heuristic knowledge like: *"IF a structure of CMOS transistors requires buffering of its output THEN implement the inverse of the function corresponding with the structure and add an invertor circuit to the output"*.

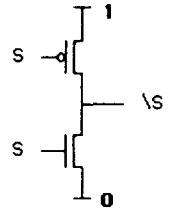


Figure 7: Structure of an inverting buffer using CMOSTs

The problem posed above may be typified as a decomposition into a transistor structure implementing the inverse of the EXOR and one for the invertor. Numerous possibilities for configuring these circuits can be thought up. A truth table of z' - the inverse of z - can serve as an intermediate description of the function belonging to the first. Transformation to the structural view gives a black-box description, representing all possible implementations. In figure 8 and 9 only two of these are shown using strong level switching logic only, respectively permitting weak levels as well. This illustrates that the number of transistors required may be largely reduced by taking advantage of the availability of a semi-open mode of operation. Also note that the first structure in figure 8 doesn't use the inverse of b thus reducing the number of transistors. Again, this knowledge should be coded, such

as to guide the search for a satisfactory implementation. Invertors are drawn as rectangles: a black-box description of the possible invertor implementations, of which the structure in figure 7 is one.

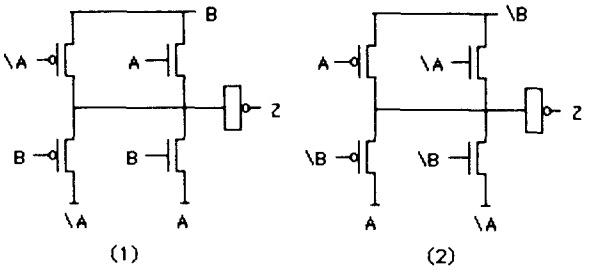


Figure 8: Two alternative structures for the EXOR based on strong-level switching logic

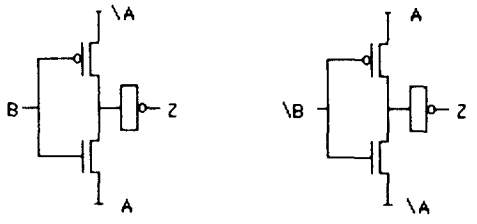


Figure 9: Two alternative structures for the EXOR based on weak-level switching logic

2.4.3 Conclusions regarding the recording method

In the context of the research described in this paper, the knowledge underlying design decisions is of primary concern. This requires the explicit recording of all major choices made during the design, even if these sometimes seem trivial to the designer. The method proposed here basically aims at relating the design actions revealed this way to the representation space.

The example illustrates how different approaches based on the same technology, may lead to totally different implementations of the EXOR. The recording method helps to determine which decisions caused the differences. A lot of the knowledge needed to find a satisfactory implementation is of a strategic nature. That is: knowledge about the way in which the design space should be travelled. For instance, strategic knowledge is involved in deciding which approach should be chosen going from the system level to the transistor level. Whether the logic properties should be strictly adhered to, or the switch-logic properties should prevail. Furthermore strategic knowledge is needed to decide when to allow for weak signal levels or when to insist on the use of strong levels only. Both largely depend on the context the implementation will be used in. So beside the design actions required, information specifying the context - like the technology used - should be provided. In terms of the vocabulary used in the model, *strategic design knowledge* concerns the sequencing of the required transformations. Given a specific goal regarding traversing part of the design space, and within a certain context, this type of knowledge specifies how to proceed.

It is also possible to extract knowledge concerning the actual carrying out of the transformations. This is called *task-specific (design) knowledge* in this paper. Like with strategic knowledge, its validity is restricted to a certain context. Consider for instance the mappings between views or levels of abstraction in the example: the actual modifications involved totally depend on the accompaning views and levels of abstraction, as well as, for instance, the design-style being 'standard cell' here. So the interrelation between the two types of knowledge is that strategic knowledge reflects an abstraction of the task-specific level, constituted by all specific, local design tasks.

3 A knowledge-based system in an open CAD-environment

To support a designer of VLSI-circuits CAD-systems have been developed, ranging from a collection of single tools for specific parts of the design trajectory, to very sophisticated CAD-environments that support the designer at every stage in the design process. We will focus on this last category, and in particular on the socalled open tool-frameworks or toolbox systems. A tool-framework consists of a collection of tools (computer-programs) for design tasks, a tool-communication protocol and a number of other provisions, like database management. Basically two approaches to the design of VLSI tool-frameworks can be distinguished; open or closed. The problem with closed, highly integrated CAD-systems is their low flexibility. In [6] it is stated that: "it is essential that a CAD system be able to support a variety of design styles and adapt easily to new developments in these areas". The authors also conclude that an open tool-framework answers these purposes best. We share this opinion; by providing the designer with alternative tools for each design task distinguished, different design styles may be supported within a single framework. Preferably these tools are based upon small, primary design steps [4], thus constituting to the maintainability of the software. This is of major importance, since the domain of VLSI-design is subject to frequent changes in design style and the available software.

In terms of the model presented in this paper tools preferably should be based upon the basic transformations as defined by the model. The smaller the tools, the easier it gets to explicitly specify the knowledge underlying the design choices. Large, integrated tools supply few opportunities for acquisition of the knowledge involved. In

terms of the administrating method knowledge concerning the application and specific properties of the tools has to do with the transformations and therefore is task-specific. Knowledge about the sequencing of tools and about 'which tools to use in which situation' is basically of a strategic nature. Both types are to be stored explicitly in the knowledge-base to assist the designer working with the CAD-system.

Because we want to adhere to the open nature of the toolbox approach, inferencing tasks should be clearly separated from actual design tasks. In many knowledge-based support systems for VLSI design built in the past both tasks were more or less intertwined. An example of such an approach is the DAA of Kowalski [7]. Like with integrated CAD-systems, the main disadvantage of this approach is the low flexibility of the resulting system. The DAA was developed for a fixed design strategy and a specific technology, making it unsuited for other design processes. In certain cases, however, integration of design tasks and inferencing system, according to a fixed strategy, may be favoured. For example, Cathedral-II [8] is intended as a silicon compiler to synthesize synchronous multiprocessor system chips for digital signal processing. Because of the dedicated nature of such a design system, integration of all tasks may be more opportune. In that case a lot of overhead - needed to obtain the open architecture of the toolbox concept - can be avoided.

The question remains whether to have a collection of separate KBS's for different parts of the design trajectory, or to have one system that allows for local customizing of its knowledge-base according to the properties of a specific stage in the design process. In both cases special provisions have to be made for the storage and use of design-strategic knowledge to support the designer with advice concerning strategic decisions.

In this context the structuring of knowledge obtained in the model is important. We have provided a uniform representation for design actions and the design itself at different stages of the design process. If the knowledge underlying these actions and decisions of strategic nature is structured according to this representation, a primairy protocol for exchanging knowledge and inferencing results is specified implicitly. Furthermore, it is necessary to distinguish between strategic design knowledge and (strategic) problem-solving knowledge. Problem-solving knowledge in general is of a meta-type, in a sense that it describes how to manipulate knowledge at the object level. Strategic design knowledge may consist of both strategic problem-solving and object-level knowledge. For example: if only a sequence of the transformations to be performed is specified, only object-level knowledge is involved.

4 Conclusions

A universally valid model of VLSI-design processes has been presented. Based on this model a method for the acquistion of design knowledge is proposed, that entails recording all major design decisions and their consequences. The model also provides a means for the structuring of the design knowledge that is acquired. The formal conceptualization of the design domain prevents frequent adaptions to the knowledge-base in the process of knowledge acquisition.

Distinguishing between task-specific knowledge - concerning the operation of design tools - and strategic knowledge - about the sequencing of tools -, it becomes possible to regard a design process as a set of strategic decisions, in terms of the available tools (or the transformations involved).

The goal is to obtain a more flexible design environment, which is capable of interactively assisting the designer in a larger number of situations. Advice may consist of strategic information, regarding when to use which tool (or perform a manual task), or information about a specific tool (like what its initial parameter-values should be).

References

[1] Gajski, D.D. & Kuhn, R.M.: 'Introduction: New VLSI tools', IEEE Computer, pp 11-14, Dec 1983

[2] Knapp, D.W. & Parker, A.C.: 'A unified representation for design information', Proc. IFIP Conf. on CHDL 7, Koomen & Moto-Oka (eds.), pp 337-353, 1985

[3] Walker, R.A. & Thomas, D.E.: 'A model of design representation and synthesis', Proc. ACM/IEEE DAC 22, pp 453-458, 1985

[4] Spaanenburg, L.: 'Structured design of digital integrated systems with distributed control', University of Twente, Dep. of Electrical Engineering, Ph.D. Thesis, 1987

[5] Huijs, C. & Spaanenburg, L.: 'Separation of hierarchies in VLSI design demonstrated on adders', University of Twente, Dep. of Computer Science, memorandum ISBN 90-365-0145 8, 1987

[6] Newton, A.R. & Sangiovanni-Vincentelli, A.L.: 'Computer-aided design for VLSI cicruits', IEEE Computer, pp 38-60, April 1986

[7] Kowalski, T.J.: 'An A.I. approach to VLSI design', Kluwer Academic Publishers, Boston, 1985

[8] Man, H. De & Rabaey,J. & Six, P. &Claesen, L. (IMEC) - 'Cathedral-II: a silicon compiler for digital signal processing'. IEEE Design & Test, pp 13-25, Dec 1986

[9] Spaanenburg, L.: 'Variations on the EXOR in static CMOS logic', University of Twente, Dep. of Electrical Engineering, internal report 87C097, 1987